Functional Programming
Reduction Strategies.

In this lab, we will experiment with evaluation strategies used in lambda calculus.

Exercițiul 0.1. Recall the functions `reduce1` and `reduce1'`:

```
data Term = Var Id
     | App Term Term
     | Lambda Id Term deriving (Show, Eq)
reduce1' :: Term -> [Id] -> Maybe Term
reduce1' (Var id') _ = Nothing
reduce1' (App (Lambda id term) term') avoid =
  Just (casubst id term' term avoid)
reduce1' (App term1 term2) avoid = case reduce1' term1 avoid of
  Nothing -> case reduce1' term2 avoid of
               Nothing -> Nothing
               Just term2' -> Just (App term1 term2')
  Just term1' -> Just (App term1' term2)
reduce1' (Lambda id term) avoid = case reduce1' term avoid of
  Nothing -> Nothing
  Just term' -> Just (Lambda id term')

reduce1 :: Term -> Maybe Term
reduce1 t = reduce1' t (vars t)
```

Questions: What is the evaluation strategy implemented above? Test the strategy on the following examples:

$$1.(\lambda x_1.x_1)\ \Big((\lambda x_2.x_2)\ (\lambda z.(\lambda x_3.x_3)\ z)\Big);$$
$$2.(\lambda x_1.\lambda x_2.x_2)\ \Big((\lambda x.x)\ (\lambda y.y)\Big).$$

Exercițiul 0.2. Starting from the implementation of the strategy from Exercise 0.1, implement the *call-by-name* (CBN) strategy. Recall that for this strategy, reductions inside a lambda-abstraction are not allowed. Test the implementation of the strategy on the examples from Exercise 0.1. Can you identify an advantage of the CBN strategy?

Test the strategy on the following example as well:

$$(\lambda x_1.x_1\ x_1)\ \Big((\lambda x.x)\ (\lambda y.y)\Big).$$

How many calculation steps are there associated with this strategy?

Exercițiul 0.3. What is the evaluation strategy implemented by the functions below?

```
strategy1' :: Term -> [Id] -> [Term]
strategy1' (Var _) _ = []
strategy1' (App (Lambda id term) term') avoid = [casubst id term' term avoid] ++
  let all = strategy1' term avoid in
  let all' = strategy1' term' avoid in
  [ App (Lambda id successorTerm) term' | successorTerm <- all ] ++
  [ App (Lambda id term) successorTerm' | successorTerm' <- all']
strategy1' (App term1 term2) avoid =
  let all1 = strategy1' term1 avoid in
  let all2 = strategy1' term2 avoid in
  [ App sterm1 term2 | sterm1 <- all1 ] ++
  [ App term1 sterm2 | sterm2 <- all2 ]
strategy1' (Lambda id term) avoid =
  let all = strategy1' term avoid in
  [ Lambda id sterm | sterm <- all ]

strategy1 :: Term -> [Term]
strategy1 term = strategy1' term (vars term)

strategy :: Term -> [Term]
strategy term = let all = strategy1 term in case all of
   [] -> [term]
   _ -> concat (map strategy all)
```

Test this strategy against the examples from previous exercises.

Exercițiul 0.4. Implement the *call-by-value* (CBV) strategy and test the implementation on the examples from the previous exercises.

Exercițiul 0.5. Compare the calculations associated with the CBV and CBN strategies when they are run over the examples:

$$1.(\lambda x_1.\lambda x_2.x_2)\ \Big((\lambda x.x)\ (\lambda y.y)\Big);$$
$$2.(\lambda x_1.x_1\ x_1)\ \Big((\lambda x.x)\ (\lambda y.y)\Big).$$

Exercițiul 0.6. Implement the Applicative Order strategy.